User Manual and What To Do
CPSC 2150
Simple (Unordered) Graphs

Gladys Monagan
Department of Computer Science
Langara College

March 25, 2023

Abstract

In order to give the CPSC 2150 a chance to concentrate on implementing the data structure *list of neighbours* graph representation so as to solve problems by implementing algorithms, I wrote a "menu style" test program and I am supplying some test files.

1 Running solver

1.1 running the program interactively

The program solver can be run interactively (as it reads from std::cin) by typing in the commands.

Run the program and on Linux and the Windows PowerShell

```
./solver
```

on Windows

```
solver.exe
```

1.2 batch mode

The other option is to pass a predefined line argument `-batch` that outputs every character read and outputs the results. Let's take as example input coming from a file `cmds.txt`.

Use input redirection with < and run the program on Linux (and MacOS) with

```
./solver -batch < cmds.txt
```

and on Windows (use CMD not the PowerShell)

```
solver.exe -batch < cmds.txt
```

For whatever reason, as far as I know, the PowerShell does not support input redirection. A sample file `cmds.txt` has been provided. However, I strongly recommend that you start testing by running the program interactively.

If you wanted to write the output into a file called `output.txt`, you could use these commands on Linux (and MacOS) with

```
./solver -batch < cmds.txt > output.txt
```

and on Windows (but not on PowerShell)

```
solver.exe -batch < cmds.txt > output.txt
```

2 Commands of the program solver

When running the program in the non-batch mode, the user can see the menu of commands

```
      ------------- choose a command --------------------------
(i) - input the file name that contains the graph
(p) - print the graph
(n) - number of vertices in the graph
(c) - determines whether the graph is connected or not
(f) - find a cycle
(l) - list the connected components (one set of vertices per line)
(a) - apath length: calculate a path length between two vertices
(t) - twin the graph by making a copy
(r) - reset (or restart) meaning done with this graph
(m) - mention or comment, the subsequent line is ignored
(q) - quit the program altogether
      ----------------------------------------------------------
```

Only the first letter of the command is used but the user could type the complete word. For instance, instead of `i`, the user can type `input`

2.1 command input

`input` is for entering the data from a file to build a graph.
The user is queried for a file name e.g *data1.txt* as provided with the assignment.
In the file *solver.cpp*, the function `inputGraph` opens the input stream, making sure that the file is readable and reports errors if needed.

2.1.1 to implement in the class Graph: the operator for input

Implement in the file *Graph.cpp* the overloaded assignment operator >>.
Read into the instance of the class Graph the information from the open input stream, building a list of neighbours data structure using a dynamic array.
Do not use the STL.
Before reading into the new graph, if the old graph (passed by reference) has not been deallocate it, deallocate it first.

2.2 command print

Print the graph: it is up to you how you want to display the graph.

2.2.1 to implement in the class Graph: the operator for output

Implement in the file *Graph.cpp* the overloaded assignment operator <<

2.3 command number

Give the number of vertices in the current graph.

2.3.1 to implement in the class Graph: numberOfVertices

2.4 command connected

Output whether the graph is connected or not.

2.4.1 to implement in the class Graph: isConnected

Determine if the graph is connected (cf. class notes as needed). You must use a **breadth first search** approach.

2.5 command find a cycle

Output whether the graph has at least one cycle or not.

2.5.1 to implement in the class Graph: hasCycle

Determine if the graph has a cycle or not.

2.6 command list the connected component

A graph can have several connected components. This command outputs *all* the connected components by listing the vertices of each connected component. Each connected component is on a separate line.

For instance, Figure 1 has 2 connected components. One connected component is the set of vertices $\{0, 3\}$ and the other component is the set of vertices $\{1, 6, 4, 5, 2\}$.

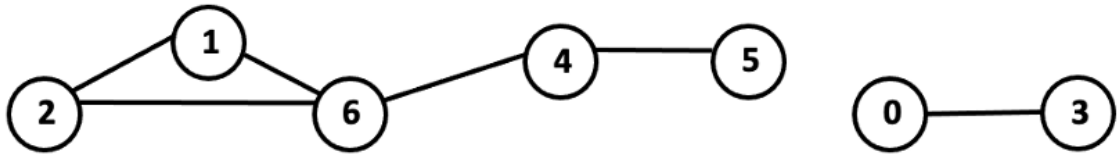

Figure 1: file data0.txt

2.6.1 to implement in the class Graph: listComponent

In the file *Graph.cpp* , the function `listComponents` that you are to implement outputs using the passed output stream.
List each set of vertices in one line.
A single vertex, without any edges adjacent to it, is a connected component and needs to be listed in a single line.

Example, not necessarily in this order, the output for Figure 1 could be

```
0 3
1 6 4 5 2
```

the order of the vertices in the one line is not important nor is there a specific order to the lines. This mean that an alternate output could be

```
1 6 4 5 2
3 0
```

2.7 command apath

Calculate the length of the path between two vertices.
The user is queried to enter the source vertex and the destination vertex.
If the source and the destination vertices are the same, the path length is 0.
If there is no path from the source vertex to the destination vertex, -1 is printed.
If the vertex entered is not a vertex of the graph, -1 is printed.

2.7.1 to implement in the class Graph: pathLength

2.8 command twin the graph

Twinning the graph simply means making a copy of a graph. The copy is manipulated with the same commands that are used for the primary graph. Exit the twin mode, with `reset`.

2.9 command reset

Indicates that the user is done with the graph.
Presumably the user wants to read another graph.
`reset` is also used to stop the twinning mode.

2.10 command mention

Mention or document or make a remark in the next line. This is useful in the input file to indicate something about the graph.
The line read after the `mention` command is echoed.

2.11 command quit

Exit `solver`

3 Format of the Input Files provided

The first integer is the number of vertices in the graph, say n. The subsequent pairs of numbers correspond to unordered edges. We don't know how many edges there are so we continue reading until the input stream enters the fail state (be it because a non numeric character is encountered or, it is the end of the file).

There are no self-loops nor are there parallel edges in our representation of a graph. The vertices are numbered 0 1 2 3 ... $n-1$.

When inserting an edge into the graph, check that both vertices of the edge are in the range of possible values. If the edge is already there, do not enter it again.

Example data1.txt

```
7
0 1
1 2
2 3
2 4
4 3
```

- there are 7 vertices in the graph
- the vertices are numbered 0 1 2 3 4 5 6
- there are 5 unordered edges in this graph: edge 0 1, edge 1 2, edge 2 3, edge 2 4, edge 4, 3
- there are two single vertices not connected to anything, namely the vertex 5 and the vertex 6

